

Spark

Screen Shots of the analysis

1. Launching an Amazon EMR cluster

The screenshot shows the AWS Management Console interface for an Amazon EMR cluster named 'SparkCluster'. The cluster is in the 'Starting' state. The left sidebar contains navigation links for Amazon EMR, EMR Studio, EMR Serverless, EMR on EC2, Clusters, Notebooks, Git repositories, Security configurations, Block public access, VPC subnets, Events, EMR on EKS, Virtual clusters, Help, and What's new. The main content area displays the cluster details, including a summary, application user interfaces, monitoring, hardware, configurations, events, steps, and bootstrap actions. The summary section shows the cluster ID (j-1MXZ0BOH4BRQ), creation date (2023-03-01 09:02 UTC+5:30), elapsed time (32 seconds), and termination protection (Off). The configuration details section shows the release label (emr-5.36.0), Hadoop distribution (Amazon), applications (Spark 2.4.8, Zeppelin 0.10.0), log URI (s3://aws-logs-055500521311-us-east-1/elasticmapreduce/), EMRFS consistent view (Disabled), custom AMI ID (None), and Amazon Linux Release (2.0.20230207.0). The network and hardware section shows the availability zone (us-east-1d), subnet ID (subnet-08c0b0b1e5aa15171), master node provisioning (1 m4.large), core node provisioning (2 m4.large), and task node provisioning (None). The security and access section shows the key name (vockey), EC2 instance profile (EMR_EC2_DefaultRole), EMR role (EMR_DefaultRole), and security groups for the master node (sg-0674cd0384300a21a).

The screenshot shows the AWS Management Console interface for the same Amazon EMR cluster 'SparkCluster', now in the 'Waiting' state. A notification banner at the top indicates that the new EMR console will become the default on Feb 28, 2023. The cluster details are updated: the elapsed time is now 6 minutes, and the master public DNS is ec2-54-242-161-131.compute-1.amazonaws.com. The application user interfaces section shows persistent user interfaces (Spark history server, YARN timeline server) and on-cluster user interfaces (Not Enabled). The network and hardware section shows the master node bootstrapping (1 m4.large) and core node provisioning (2 m4.large). The security and access section remains the same.

2. Create an S3 Bucket and Upload data and CSV file to S3 bucket

The screenshot shows the AWS Management Console interface. At the top, a green notification banner states: "Successfully created bucket 'sparkclusterassignment1'". Below this, the "Buckets" page is displayed. The left sidebar contains navigation links for "Buckets", "Access Points", "Object Lambda Access Points", "Multi-Region Access Points", "Batch Operations", "IAM Access Analyzer for S3", "Storage Lens", "Dashboards", "AWS Organizations settings", "Feature spotlight", and "AWS Marketplace for S3". The main content area shows an "Account snapshot" with metrics: Total storage (3.0 MB), Object count (394), and Average object size (7.7 KB). Below this, a table lists the buckets:


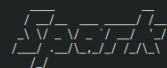
Name	AWS Region	Access	Creation date
aws-logs-055500521311-us-east-1	US East (N. Virginia) us-east-1	Objects can be public	January 26, 2023, 18:11:58 (UTC+05:30)
sparkclusterassignment1	US East (N. Virginia) us-east-1	Bucket and objects not public	March 1, 2023, 09:04:33 (UTC+05:30)

The screenshot shows the "Upload: status" page in the AWS Management Console. A green notification banner at the top states: "Upload succeeded". The page displays the upload status for the bucket "sparkclusterassignment1". The "Summary" section shows the destination "s3://sparkclusterassignment1/input/" and the upload status: "Succeeded" (1 file, 99.6 KB, 100.00%) and "Failed" (0 files, 0 B, 0%). Below this, the "Files and folders" section shows a table with the following columns: Name, Folder, Type, Size, Status, and Error. The table contains one entry: "1 file, 99.6 KB".

3. Connecting to the Hadoop main node by using SSH

[illegible]

4. Running spark

```
hadoop@ip-172-31-30-218:~  
Last login: Wed Mar  1 03:50:01 2023  
 Amazon Linux 2 AMI  
https://aws.amazon.com/amazon-linux-2/  
2 package(s) needed for security, out of 8 available  
Run 'sudo yum update' to apply all updates.  
  
EEEEEEEEEEEEEEEEEEEE MWWWWWWW MWWWWWWW RRRRRRRRRRRRRRRR  
E:EEEEEEEEEEEEEEEEEEEE M:EEEEEEEE M:EEEEEEEE R:EEEEEEEEEEEE  
EE:EEEEEEEEEEEEEEEEEEEE M:EEEEEEEE M:EEEEEEEE R:RRRRRRRRRRRR  
E:EE EEEEE M:EEEEEEEE M:EEEEEEEE RR:RR R:RR  
E:EE EEEEE M:EEEEEEEE M:EEEEEEEE R:RR R:RR  
E:EEEEEEEEEEEEEEEEEEEE M:EEEEEEEE M:EEEEEEEE R:RRRRRRRRRRRR  
E:EEEEEEEEEEEEEEEEEEEE M:EEEEEEEE M:EEEEEEEE R:EEEEEEEEEEEE  
E:EEEEEEEEEEEEEEEEEEEE M:EEEEEEEE M:EEEEEEEE R:RRRRRRRRRRRR  
E:EE EEEEE M:EEEEEEEE M:EEEEEEEE R:RR R:RR  
EE:EEEEEEEEEEEEEEEEEEEE M:EEEEEEEE M:EEEEEEEE R:RR R:RR  
E:EEEEEEEEEEEEEEEEEEEE M:EEEEEEEE M:EEEEEEEE RR:RR R:RR  
EEEEEEEEEEEEEEEEEEEE MWWWWWWW MWWWWWWW RRRRRRRRRRRRRRRR  
  
hadoop@ip-172-31-30-218 ~]$ spark-shell  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
23/03/01 03:53:13 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.  
spark context Web UI available at http://ip-172-31-30-218.ec2.internal:4040  
spark context available as 'sc' (master = yarn, app id = application_1677641815419_0002).  
spark session available as 'spark'.  
Welcome to  
 version 2.4.8-azn-2  
  
Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_362)  
Type in expressions to have them evaluated.  
Type :help for more information.  
  
scala>
```

5. Imports and initializing the spark.

```
hadoop@ip-172-31-30-218:~  
EEEEEEEEEEEEEEEEEEEE WWWWWW          WWWWWW RRRRRRRRRRRRRR  
E:::EEEEEEEEEEEEEEEE E M:::MM          M:::MM R:::RRRRRRRRRRR  
E:::EEEEEEEEEEEEEEEE M:::MM          M:::MM R:::RRRRRRRRRRR  
E:::E EEEEE M:::MM          M:::MM RRRRRRRRRRRR R:::R  
E:::E M:::MM M:::MM M:::MM R:::R R:::R  
E:::EEEEEEEEEEEE M:::MM M:::MM M:::MM R:::RRRRRRRRRRR  
E:::EEEEEEEEEEEE M:::MM M:::MM M:::MM R:::RRRRRRRRRRR  
E:::E M:::MM M:::MM R:::R R:::R  
E:::E EEEEE M:::MM M:::MM M:::MM R:::R R:::R  
E:::EEEEEEEEEEEE M:::MM M:::MM R:::R R:::R  
E:::EEEEEEEEEEEE M:::MM M:::MM RRRRRRRRRRRR R:::R  
EEEEEEEEEEEEEEEEEEEE WWWWWW          WWWWWW RRRRRRRRRRRR  
hadoop@ip-172-31-30-218 ~$ spark-shell  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
23/03/01 03:53:13 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.  
Spark context web UI available at http://ip-172-31-30-218.ec2.internal:4000  
Spark context available as 'sc' (master = yarn, app id = application_1677641815419_0002).  
Spark session available as 'spark'.  
Welcome to  
 version 2.4.8-anzn-2  
Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_362)  
Type in expressions to have them evaluated.  
Type :help for more information.  
scala> import org.apache.spark.sql.SparkSession  
import org.apache.spark.sql.SparkSession  
scala> val spark = SparkSession  
spark: org.apache.spark.sql.SparkSession.type = org.apache.spark.sql.SparkSession$076d4e1af  
scala> val spark = SparkSession.builder.appName("SparkTest").getOrCreate();  
23/03/01 03:54:31 WARN SparkSessionBuilder: Using an existing SparkSession; some spark core configurations may not take effect.  
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession$59c75b72  
scala>
```

6. Load data and create a view

```
hadoop@ip-172-31-30-218:~  
E::::E      EEEE M::::M      MMM      M::::M      R:::R      R::::R  
EE:::::EEEEEEEEE:::E M::::M      M::::M      R:::R      R::::R  
E:::::::::::E M::::M      M::::M      R:::R      R::::R  
EEEEEEEEEEEEEEEE MMMMMM      MMMMMM      RRRRRRR      RRRRRR  
  
hadoop@ip-172-31-30-218 ~]$ spark-shell  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
23/03/01 03:53:13 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.  
Spark context Web UI available at http://ip-172-31-30-218.ec2.internal:4040  
Spark context available as 'sc' (master = yarn, app id = application_1677641815419_0002).  
Spark session available as 'spark'.  
Welcome to  
  
      _/ _ \| | | |  
     / ___ \| |_| |  
    / /___ \| | | |  
   /_____\|_|_|_|  
  version 2.4.8-amzn-2  
  
Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_362)  
Type in expressions to have them evaluated.  
Type :help for more information.  
  
scala> import org.apache.spark.sql.SparkSession  
import org.apache.spark.sql.SparkSession  
  
scala> val spark = SparkSession  
spark: org.apache.spark.sql.SparkSession.type = org.apache.spark.sql.SparkSession$@76d4e1af  
  
scala> val spark = SparkSession.builder.appName("SparkTest").getOrCreate();  
23/03/01 03:54:31 WARN SparkSessionBuilder: Using an existing SparkSession; some spark core configurations may not take effect.  
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession$@59c75b72  
  
scala> val csvFile = "s3://sparkclusterassignment1/input/DelayedFlights-updated.csv"  
csvFile: String = s3://sparkclusterassignment1/input/DelayedFlights-updated.csv  
  
scala> val df = spark.read.format("csv").option("inferSchema", "true").option("header", "true").load(csvFile);  
df: org.apache.spark.sql.DataFrame = [_c0: int, Year: int ... 28 more fields]  
  
scala> df.createOrReplaceTempView("delay_flights");  
23/03/01 03:56:39 WARN Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' in SparkEnv.conf.  
scala> []
```

7. Querying and analyzing the resulting dataset

```
csvFile: String = S3://sparkcluster-assignment1/output/delayedflights-updated.csv

scala> val df = spark.read.format("csv").option("inferSchema", "true").option("header", "true").load(csvFile);
df: org.apache.spark.sql.DataFrame = [_c0: int, Year: int ... 28 more fields]

scala> df.createOrReplaceTempView("delay_flights");
23/03/01 03:56:39 WARN Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' in SparkEnv.conf.

scala> spark.time(spark.sql("SELECT Year, avg((CarrierDelay / ArrDelay)*100) from delay_flights GROUP BY Year").show())
23/03/01 03:57:30 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException

Year|avg((((CAST(CarrierDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|24.557549755575373|
2007|19.850067017971283|
2010|21.89310246015957|
2006|30.453296261292596|
2004|43.64459443230066|
2005|28.01977637202288|
2009|28.33058554239575|
2008|28.88346981456985|
-----+-----+
```

8. Other results

```
scala> spark.time(spark.sql("SELECT Year, avg((CarrierDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(CarrierDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|24.557549755575373|
2007|19.850007017971283|
2010|21.89310246015957|
2006|30.453296261292596|
2004|43.64459443230066|
2005|28.01977637202288|
2009|28.33058554239575|
2008|28.88346981456985|
-----+-----+
Time taken: 863 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((NASDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(NASDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|29.686276314267346|
2007|30.625925917941924|
2010|33.87351363404217|
2006|18.119312329937703|
2004|18.24570061769958|
2005|16.63868805373129|
2009|37.63093336628511|
2008|30.16552562594132|
-----+-----+
Time taken: 832 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((WeatherDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(WeatherDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|7.8319479664511205|
2007|4.042975783210287|
2010|2.9023312955584664|
2006|4.588604183967953|
2004|6.4475279976916555|
2005|5.85069715149616|
2009|0.453161615197982363|
2008|3.7254490054008955|
-----+-----+
Time taken: 814 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((LateAircraftDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(LateAircraftDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|37.924225963706164|
2007|45.252432744291134|
2010|41.331052610239794|
2006|46.838787224801735|
2004|31.662176952308105|
2005|49.490838422749654|
2009|33.585314999939314|
2008|37.22555555408794|
-----+-----+
Time taken: 977 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((SecurityDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(SecurityDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|0.0|
2007|0.22865853658536586|
2010|0.0|
2006|0.0|
2004|0.0|
2005|0.0|
2009|0.0|
2008|0.0|
-----+-----+
Time taken: 787 ms
```

Year wise carrier delay from 2003-2010


```
scala> spark.time(spark.sql("SELECT Year, avg((CarrierDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(CarrierDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|                                                                 24.557549755575373|
2007|                                                                 19.850007017971283|
2010|                                                                 21.89310246015957|
2006|                                                                 30.453296261292596|
2004|                                                                 43.64459443230066|
2005|                                                                 28.01977637202288|
2009|                                                                 28.33058554239575|
2008|                                                                 28.88346981456985|
-----+-----+

Time taken: 732 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((CarrierDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(CarrierDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|                                                                 24.557549755575373|
2007|                                                                 19.850007017971283|
2010|                                                                 21.89310246015957|
2006|                                                                 30.453296261292596|
2004|                                                                 43.64459443230066|
2005|                                                                 28.01977637202288|
2009|                                                                 28.33058554239575|
2008|                                                                 28.88346981456985|
-----+-----+

Time taken: 662 ms
scala> █
```

```
scala> spark.time(spark.sql("SELECT Year, avg((CarrierDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(CarrierDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|                                                                 24.557549755575373|
2007|                                                                 19.850007017971283|
2010|                                                                 21.89310246015957|
2006|                                                                 30.453296261292596|
2004|                                                                 43.64459443230066|
2005|                                                                 28.01977637202288|
2009|                                                                 28.33058554239575|
2008|                                                                 28.88346981456985|
-----+-----+

Time taken: 808 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((CarrierDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(CarrierDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|                                                                 24.557549755575373|
2007|                                                                 19.850007017971283|
2010|                                                                 21.89310246015957|
2006|                                                                 30.453296261292596|
2004|                                                                 43.64459443230066|
2005|                                                                 28.01977637202288|
2009|                                                                 28.33058554239575|
2008|                                                                 28.88346981456985|
-----+-----+

Time taken: 643 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((CarrierDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show())
-----+-----+
Year|avg((((CAST(CarrierDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|                                                                 24.557549755575373|
2007|                                                                 19.850007017971283|
2010|                                                                 21.89310246015957|
2006|                                                                 30.453296261292596|
2004|                                                                 43.64459443230066|
2005|                                                                 28.01977637202288|
2009|                                                                 28.33058554239575|
2008|                                                                 28.88346981456985|
-----+-----+

Time taken: 834 ms
scala> █
```

Year wise NAS delay from 2003-2010

```
scala> spark.time(spark.sql("SELECT Year, avg((NASDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
-----+-----+
Year|avg((((CAST(NASDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
-----+-----+
2003|                                                                 29.686276314267346|
2007|                                                                 30.625925917941924|
2010|                                                                 33.07351363404217|
2006|                                                                 18.119312329937703|
2004|                                                                 18.24570061769958|
2005|                                                                 16.63868805373129|
2009|                                                                 37.63093336628511|
2008|                                                                 30.16552562594132|
-----+-----+

Time taken: 813 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((NASDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+
|Year|avg(((CAST(NASDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+
|2003|29.686276314267346|
|2007|30.625925917941924|
|2010|33.87351363404217|
|2006|18.119312329937703|
|2004|18.24570061769958|
|2005|16.63868805373129|
|2009|37.63093330628511|
|2008|30.16552562594132|
+-----+

Time taken: 764 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((NASDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+
|Year|avg(((CAST(NASDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+
|2003|29.686276314267346|
|2007|30.625925917941924|
|2010|33.87351363404217|
|2006|18.119312329937703|
|2004|18.24570061769958|
|2005|16.63868805373129|
|2009|37.63093330628511|
|2008|30.16552562594132|
+-----+

Time taken: 610 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((NASDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+
|Year|avg(((CAST(NASDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+
|2003|29.686276314267346|
|2007|30.625925917941924|
|2010|33.87351363404217|
|2006|18.119312329937703|
|2004|18.24570061769958|
|2005|16.63868805373129|
|2009|37.63093330628511|
|2008|30.16552562594132|
+-----+

Time taken: 815 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((NASDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+
|Year|avg(((CAST(NASDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+
|2003|29.686276314267346|
|2007|30.625925917941924|
|2010|33.87351363404217|
|2006|18.119312329937703|
|2004|18.24570061769958|
|2005|16.63868805373129|
|2009|37.63093330628511|
|2008|30.16552562594132|
+-----+

Time taken: 590 ms
```

Year wise Weather delay from 2003-2010

```
scala> spark.time(spark.sql("SELECT Year, avg((WeatherDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+
|Year|avg(((CAST(WeatherDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+
|2003|7.8319479664511205|
|2007|4.042975783210287|
|2010|2.9023312955584664|
|2006|4.588604183967953|
|2004|6.4475279976916555|
|2005|5.85069715149616|
|2009|0.45316615137982363|
|2008|3.7254490054008955|
+-----+

Time taken: 773 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((WeatherDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+
|Year|avg(((CAST(WeatherDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+
|2003|7.8319479664511205|
|2007|4.042975783210287|
|2010|2.9023312955584664|
|2006|4.588604183967953|
|2004|6.4475279976916555|
|2005|5.85069715149616|
|2009|0.45316615137982363|
|2008|3.7254490054008955|
+-----+

Time taken: 689 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((WeatherDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg(((CAST(WeatherDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+-----+
|2003|7.8319479664511205|
|2007|4.042975783210287|
|2010|2.9023312955584664|
|2006|4.588604183967953|
|2004|6.4475279976916555|
|2005|5.85069715149616|
|2009|0.45316615137982363|
|2008|3.7254490054008955|
+-----+-----+

Time taken: 735 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((WeatherDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg(((CAST(WeatherDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+-----+
|2003|7.8319479664511205|
|2007|4.042975783210287|
|2010|2.9023312955584664|
|2006|4.588604183967953|
|2004|6.4475279976916555|
|2005|5.85069715149616|
|2009|0.45316615137982363|
|2008|3.7254490054008955|
+-----+-----+

Time taken: 743 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((WeatherDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg(((CAST(WeatherDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+-----+
|2003|7.8319479664511205|
|2007|4.042975783210287|
|2010|2.9023312955584664|
|2006|4.588604183967953|
|2004|6.4475279976916555|
|2005|5.85069715149616|
|2009|0.45316615137982363|
|2008|3.7254490054008955|
+-----+-----+

Time taken: 771 ms
scala>
```

Year wise late aircraft delay from 2003-2010

```
scala> spark.time(spark.sql("SELECT Year, avg((LateAircraftDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg(((CAST(LateAircraftDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+-----+
|2003|37.924225963706164|
|2007|45.252432744291134|
|2010|41.331052610239794|
|2006|46.838787224801735|
|2004|31.662176952308105|
|2005|49.490838422749654|
|2009|33.585314999939314|
|2008|37.22555555408794|
+-----+-----+

Time taken: 589 ms
scala>
```

```
scala> spark.time(spark.sql("SELECT Year, avg((LateAircraftDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg(((CAST(LateAircraftDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+-----+
|2003|37.924225963706164|
|2007|45.252432744291134|
|2010|41.331052610239794|
|2006|46.838787224801735|
|2004|31.662176952308105|
|2005|49.490838422749654|
|2009|33.585314999939314|
|2008|37.22555555408794|
+-----+-----+

Time taken: 583 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((LateAircraftDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg(((CAST(LateAircraftDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE)))|
+-----+-----+
|2003|37.924225963706164|
|2007|45.252432744291134|
|2010|41.331052610239794|
|2006|46.838787224801735|
|2004|31.662176952308105|
|2005|49.490838422749654|
|2009|33.585314999939314|
|2008|37.22555555408794|
+-----+-----+

Time taken: 611 ms
scala>
```

```
scala> spark.time(spark.sql("SELECT Year, avg((LateAircraftDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg((((CAST(LateAircraftDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
+-----+-----+
|2003|37.924225963706164|
|2007|45.252432744291134|
|2010|41.331052610239794|
|2006|46.838787224801735|
|2004|31.662176952308105|
|2005|49.490838422749654|
|2009|33.585314999939314|
|2008|37.22555555408794|
+-----+-----+

Time taken: 538 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((LateAircraftDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg((((CAST(LateAircraftDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
+-----+-----+
|2003|37.924225963706164|
|2007|45.252432744291134|
|2010|41.331052610239794|
|2006|46.838787224801735|
|2004|31.662176952308105|
|2005|49.490838422749654|
|2009|33.585314999939314|
|2008|37.22555555408794|
+-----+-----+

Time taken: 727 ms

scala> 
```

Year wise security delay from 2003-2010

```
scala> spark.time(spark.sql("SELECT Year, avg((SecurityDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg((((CAST(SecurityDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
+-----+-----+
|2003|0.0|
|2007|0.22865853658536586|
|2010|0.0|
|2006|0.0|
|2004|0.0|
|2005|0.0|
|2009|0.0|
|2008|0.0|
+-----+-----+

Time taken: 476 ms

scala> 
```

```
scala> spark.time(spark.sql("SELECT Year, avg((SecurityDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg((((CAST(SecurityDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
+-----+-----+
|2003|0.0|
|2007|0.22865853658536586|
|2010|0.0|
|2006|0.0|
|2004|0.0|
|2005|0.0|
|2009|0.0|
|2008|0.0|
+-----+-----+

Time taken: 812 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((SecurityDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg((((CAST(SecurityDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
+-----+-----+
|2003|0.0|
|2007|0.22865853658536586|
|2010|0.0|
|2006|0.0|
|2004|0.0|
|2005|0.0|
|2009|0.0|
|2008|0.0|
+-----+-----+

Time taken: 714 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((SecurityDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
+-----+-----+
|Year|avg((((CAST(SecurityDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))|
+-----+-----+
|2003|0.0|
|2007|0.22865853658536586|
|2010|0.0|
|2006|0.0|
|2004|0.0|
|2005|0.0|
|2009|0.0|
|2008|0.0|
+-----+-----+

Time taken: 551 ms
```

```
scala> spark.time(spark.sql("SELECT Year, avg((SecurityDelay /ArrDelay)*100) from delay_flights WHERE Year is NOT NULL GROUP BY Year").show());
-----+-----+
[Year|avg((((CAST(SecurityDelay AS DOUBLE) / CAST(ArrDelay AS DOUBLE)) * CAST(100 AS DOUBLE))))]
-----+-----+
[2003|                                                                0.0]
[2007|                                                                0.0]
[2007|                                0.22865853658536586]
[2010|                                                                0.0]
[2006|                                                                0.0]
[2004|                                                                0.0]
[2005|                                                                0.0]
[2009|                                                                0.0]
[2008|                                                                0.0]
-----+-----+
Time taken: 755 ms
scala> 
```

9. Terminating the Cluster and cleaning the S3 bucket

The screenshot displays the AWS EMR console interface. At the top, a notification states: "The new EMR console will become the default console on Feb 28, 2023. Switch to the new console. If you want, you can still switch back. Learn more". The left sidebar shows the navigation menu with options like Amazon EMR, EMR Studio, EMR Serverless, EMR on EC2, Clusters, Notebooks, Git repositories, Security configurations, Block public access, VPC subnets, Events, EMR on EKS, Virtual clusters, Help, and What's new.

The main content area shows the details for a cluster named "SparkCluster" in a "Terminating" state. The cluster was terminated by a user request. The "Summary" tab is selected, displaying the following information:

- ID:** j-1MXZ0BOH48ROQ
- Creation date:** 2023-03-01 09:02 (UTC+5:30)
- Elapsed time:** 38 minutes
- After last step completes:** Cluster waits
- Termination protection:** Off
- Tags:** --
- Master public DNS:** ec2-54-242-161-131.compute-1.amazonaws.com
- Master public DNS:** Connect to the Master Node Using SSH

The "Configuration details" tab shows the following information:

- Release label:** emr-5.36.0
- Hadoop distribution:** Amazon
- Applications:** Spark 2.4.8, Zeppelin 0.10.0
- Log URI:** s3://aws-logs-055500521311-us-east-1/elasticmapreduce/
- EMRFS consistent view:** Disabled
- Custom AMI ID:** --
- Amazon Linux Release:** 2.0.20230207.0

The "Application user interfaces" tab shows the following information:

- Persistent user interfaces:** Spark history server, YARN timeline server
- On-cluster user interfaces:** --

The "Network and hardware" tab shows the following information:

- Availability zone:** us-east-1d
- Subnet ID:** subnet-08c0db0b1e5aa15171
- Master:** Terminating 1 m4.large
- Core:** Terminating 2 m4.large
- Task:** --
- Cluster scaling:** Not enabled
- Auto-termination:** Terminate if idle for 1 hour

The "Security and access" tab is also visible.

Below the console screenshot, a green notification banner states: "Successfully emptied bucket 'sparkclusterassignment1'. View details below. If you want to delete this bucket, use the delete bucket configuration." Below this, the "Empty bucket: status" dialog is shown. It contains a summary table with the following data:

Source	Successfully deleted	Failed to delete
s3://sparkclusterassignment1	3 objects, 99.6 KB	0 objects

The "Failed to delete (0)" section shows a search bar and a table with the following columns: Name, Prefix, Version ID, Type, Last modified, Size, and Error. The table is currently empty, and a message "No failed object deletions" is displayed.

Services

Search

[Alt+S]

Global

voclabs/user2377842=munizima.23@uom.lk @ 0555-0052-1311

Amazon S3

Buckets

Access Points

Object Lambda Access Points

Multi-Region Access Points

Batch Operations

IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens

Dashboards

AWS Organizations settings

Feature spotlight

AWS Marketplace for S3

Successfully deleted bucket "sparkclusterassignment1"

Amazon S3

Buckets

Account snapshot

Last updated: Feb 27, 2023 by Storage Lens. Metrics are generated every 24 hours. [Learn more](#)

Total storage

Object count

Average object size

You can enable advanced metrics in the "default-account-dashboard" configuration.

Buckets (1)

Info

Find buckets by name

Copy ARN

Empty

Delete

Create bucket

Name	AWS Region	Access	Creation date
aws-logs-055500521311-us-east-1	US East (N. Virginia) us-east-1	Objects can be public	January 26, 2023, 18:11:58 (UTC+05:30)

Feedback

Language

© 2023, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)